USING LABVIEW FOR TELEMETRY EMULATION, ANALYSIS AND DISPLAY

by

George Wells, Member of 'Technical Staff
Edmund C. Baroth, Ph.D., Manager of the Measurement Technology Center,
Jet Propulsion Laboratory, California Institute of Technology

Abstract

An example. of a telemetry monitoring and display application using two Macintosh computers and a visual programming language (National Instruments' LabVIEW) will be discussed. One computer was used as the telemetry source and the other as (he analyzer. The telemetry stream was emulated using interface boards. User interface panels and programs will be presented. The programs will also be discussed as to case. of creation and modification. The purpose of this paper is to strew how visual-based programming can be used for advanced data analysis applications such as telemetry analysis and display, as well as for emulation of a telemetry stream.

This application was created as the result of parallel development between a visual programming team and a text-based ('C') programming team, Although not a scientific study, it was a fair comparison between different development methods and tools. With approximately eight weeks of funding over a period of three months, the visual programming effort was significantly more advanced, having gone wcii beyond the original requirements than the 'C' development effort, which did not complete the original requirements. This application verified that using visual programming can significantly reduce software development time. As a result of this initial effort, additional folio w-on work was awarded to the visual programming team.

The Measurement Technology Center has consistently achieved a reduction in software/system development time by at least a factor of four, anti up to an order of magnitude, compared to text-hasc(i software tools tailored specifically to the test and measurement environment. The most dramatic gains in productivity arc attributed to the communication among the customer, developer, and computer that arc facilitated by the visual syntax of the tools.

Introduction

The Measurement Technology Center (MTC) evaluates commercial data acquisition and analysis hardware and software products for the JPL engineering and scientific community.[1] The MTC specifically configures and delivers turn-key measurement systems that include software, user interface, sensors (e.g., thcr mocouples, pressure transducers) and signal conditioning, plus data acquisition, analysis, display, simulation and control capabilities.[2,3]

Visual programming tools arc frequently used to simplify development (compared to text-based programming) of such systems. Employment of visual programming tools that control off-the-shcif interface cards has been the most important factor in reducing time and cost of configuring these systems. Understanding the experimenter's needs is aiso critical as is an interactive approach to user interface construction anti training of the operators.

The MTC configures measurement systems based around IBM-compatible and Apple Macintosh personal computers, plus Hewlett-Packard and Sun Microsystems workstations. The MTC consistently achieves a reduction in software/systcnl development time. by at least a factor of four, anti up to an order of magnitude, compared to text-based software, tools tailored specifically to our environment.[4,5,6] Others in industry arc reporting similar increases in productivity and reduction in software/sys(crn development time and cost.[7,8,9]

The MTC was approached to create a telemetry analyzer, using the Macintosh and LabVIEW. A similar task was given to another group using text-based coding (in 'C') on a Sun workstation. Both groups were given equal time and funding levels. Although not a scientific study, it was a fair comparison between different development methods and tools. The purpose was to determine if the LabVIEW software environment was up to the task of telemetry stream decommutation and decoding and to verify advantages of visual compared to text-based programming.

The most important advantage is the increase in productivity, which can be attributed to the improved communication among the customer, developer, and computer that is facilitated by the visual syntax of the tools. Use of visual programming encourages a much more interactive process between user and programmer. Coding usually is implemented intro-actively with the customer and developer together at the computer. This reduces the time to produce both code and the number of iterations to satisfy the user, Modifications can be incorporated much easier than using text-based code.

An additional advantage is the ability to collapse code using many icons to form a single icon, which allows applications to be created which are both custom and modular. These advantages add up to an environment that can reduce software development by at least a factor of four and Up to an order of magnitude. The costs for system configuration, documentation, training and operating can also be substantially reduced.

'f'he real capability of visual programming, however, is the ability to go from conception to simulation of components, sub-systems and systems, to testing of actual hardware and control functions using a single software environment (on multiple platforms). Modules or icons that represent simulations of instruments, processes or algorithms can be easily replaced with (he actual instruments or components when they become available. That potential was demonstrated in this application, as first the telemetry generator and analyzer were simulated in one computer, then were split to emulate the actual telemetry stream.

Galileo Mission Telemetry Analyzer

The MTC is currently supporting the redesign of the computer data system for NASA's Galileo mission to Jupiter. LabVIEW is being used to assist in the ground test of the flight software redesign as one of a series of tools used to configure measurement systems.[10]

Two Macintosh computers were used, a Mac IIfx and a Quadra 950. The Mac IIfx was used to generate the telemetry stream using an interface board's digital to analog channels. The Quadra 950 functioned as the telemetry analyzer using the analog to digital converter on a similar board to capture a data channel triggered by a clock channel.

The original requirement and expectation for the task was to usc one computer for simulation and analysis of the telemetry channel (using global variables) but because that requirement was met long before the deadline, the actual generation of the telemetry channel was accomplished. The separation of the generator and analyzer into two separate computer systems was performed to approximate more closely the real environment and to allow more precise measurement of performance parameters. The telemetry data stream consists of a two-line serial interface (data and clock). Although it only needed to operate at 200 bits per second, it was tested at up to 5000 bits per second to measure the CPU margin.

Telemetry Generator

There was no actual requirement for a telemetry generator, only an analyzer. The generator was created to test the analyzer. Figure 1 is the Telemetry Generator Sequence. The source of the data is pre-determined 'Predict Tables' containing random bytes. For each instrument there is a separate table. These tables arc stored in both the generator and the analyzer computers. The overall approach is that each instrument scads packets of up to 220 bytes at random intervals of time and eventually these packets arc picked up by the analyzer and verified in its Predict Tables. Packets that are received out of sequence or arc not present in the Predict Tables will register as errors. It is a verification of the transmitting and receiving process, not the data itself.

The entire telemetry scheme can be thought of as having an independent channel for each instrument. Packets from an instrument arc not actually sent until enough arc received to fill a segment of 223 bytes (including some overhead). A Reed-Solon~on error-correcting code. of 32 bytes is appended to each segment and when eight segments arc assembled (not necessarily from the same instrument), an eight-byte PN (pseudo-nurabcr) Sync word is attached. The bytes arc sent starting with the PN Sync word and continue through the first byte of each segment followed by successive bytes of each segment. Before being sent over the telemetry channel, the stream is run through a convolution algorithm that provides additional mor-correcting capabilities but doubles the number of bits.

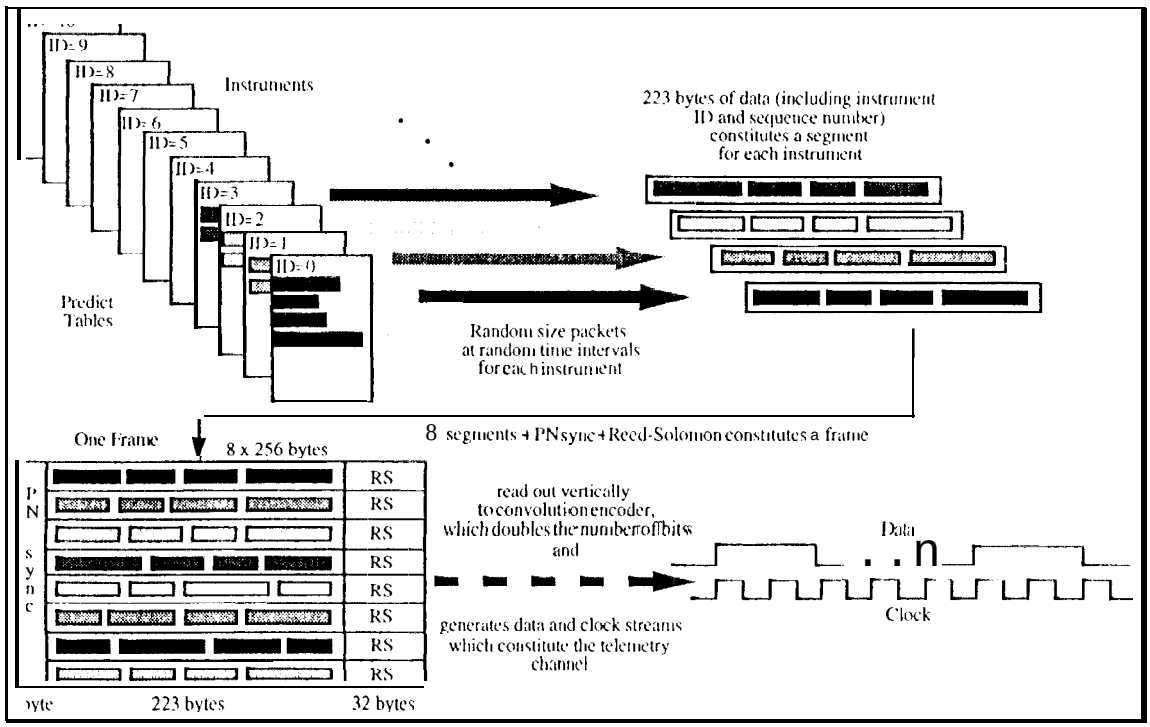

Figure 1. Telemetry Generator Sequence

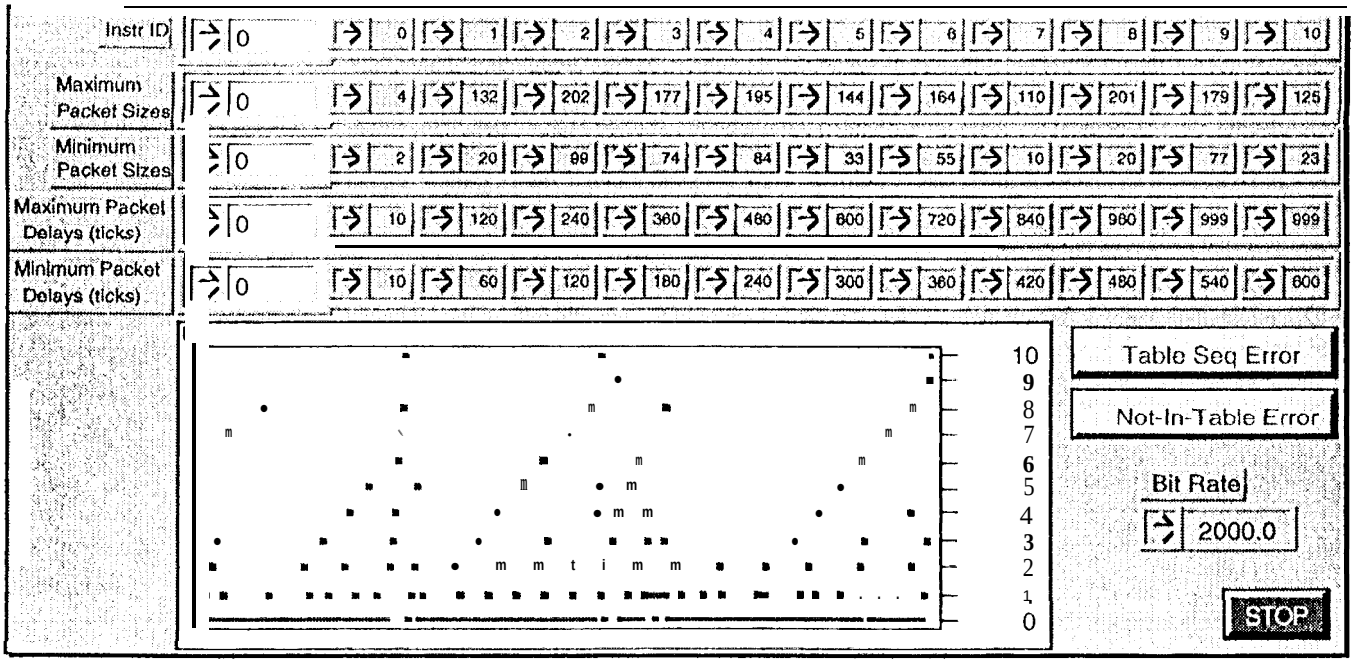

Figure 2. Telemetry Generator User Interface (Example of LabVIEW Front Panel)

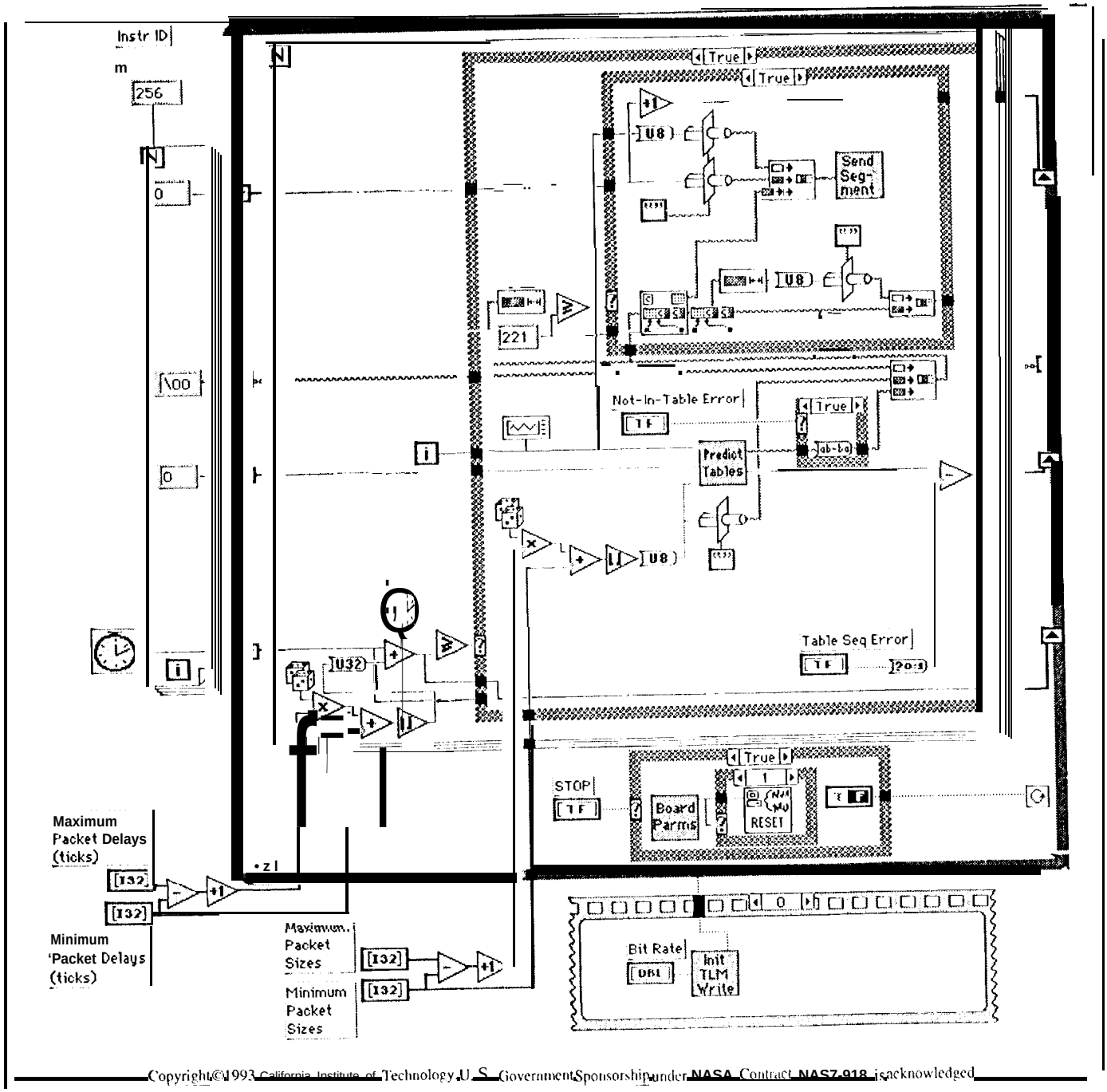

Figure 3. Telemetry Generator Program (Example of LabVIEW Diagram)

The implementation of the various algorithms (e.g., Reed-Solomon and Convolutional Encoders) has traditionally been done in hardware using shift-registers, ex-OR gates and counters. The close analogy of LabVIEW's icons to actual circuitry enabled easy and straight forward implementation of otherwise complex coding.

Figure 2 is the user interface (LabVIEW Front Panel) for the telemetry generator. The range of packet sizes and time intervals can be specified for each instrument as well as the datarate. As each packet is generated, it is displayed on the strip chart as a dot. For example, instrument (Instr ID) = 0 attempts to output a packet of between 2 and 4 bytes every 1/6 of a second (1 tick = 1/60 second), while ID = 1 attempts to output packets containing between 20 and 132 bytes every 1 to 2 seconds. These rates are adjusted by simply clicking on the up or down arrows, but are limited by the Bit Rate set on the front panel. Two types of errors can also be created to test the (diagnostic capabilities of the analyzer. They are initiated by clicking the buttons.

Figure 3 is the actual program (LabVIEW diagram) for the telemetry generator. It is responsible for temporarily storing the packets from each instrument until enough are available for a segment, at which time the 'Send Segment' VI (Virtual Instrument) is executed. This VI eventually calls the 'Calculate Reed Solomon' VI shown in Figures 4 & 5 that implements all the circuitry shown in Figure 6, but in a more general sense, in that the shift registers are put into a loop. The coefficients and RS Table are permanently stored as default values on the front panel. Notice how compact the block diagram is compared to the schematic that it emulates.

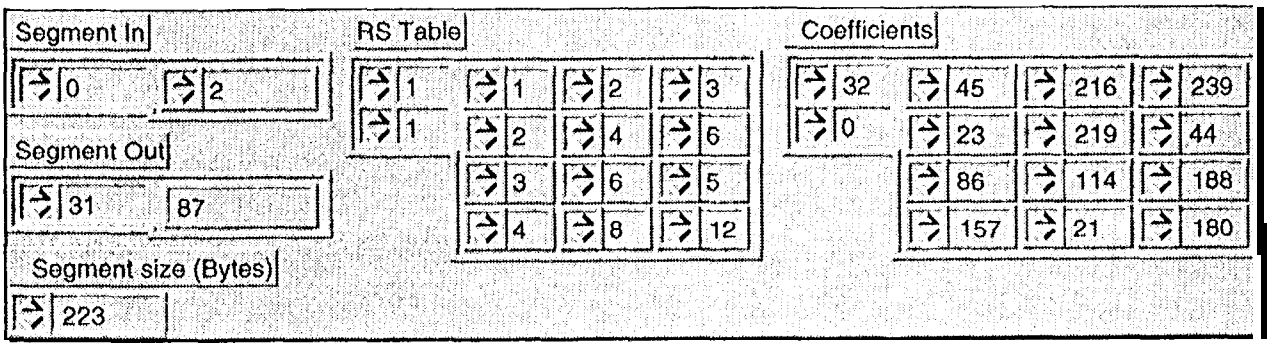

Figure 4. Calculate Reed-Solomon Front Panel

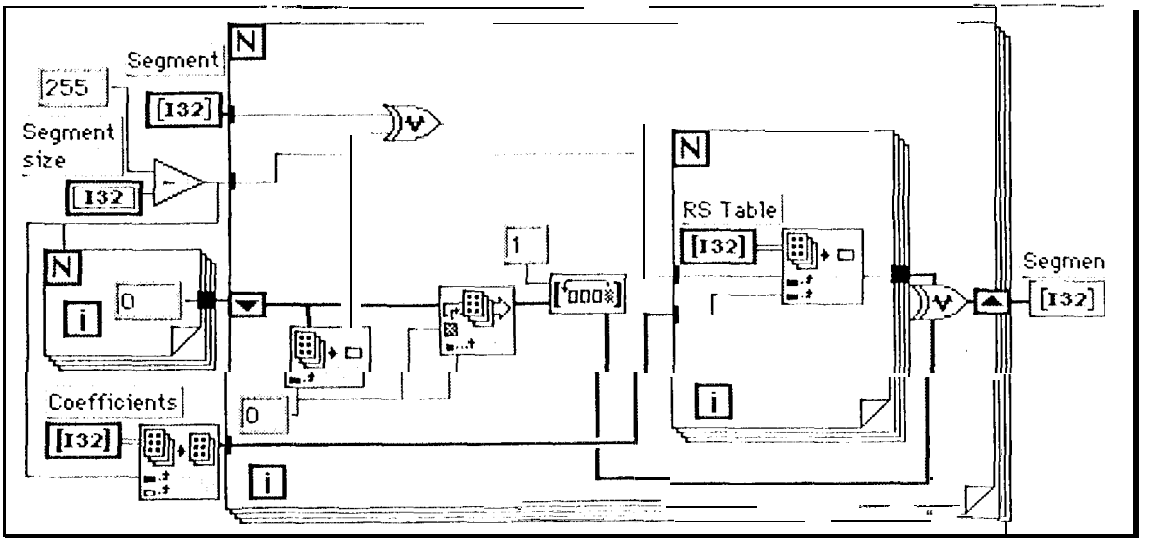

Figure 5. Calculate Reed-Solomon Diagram

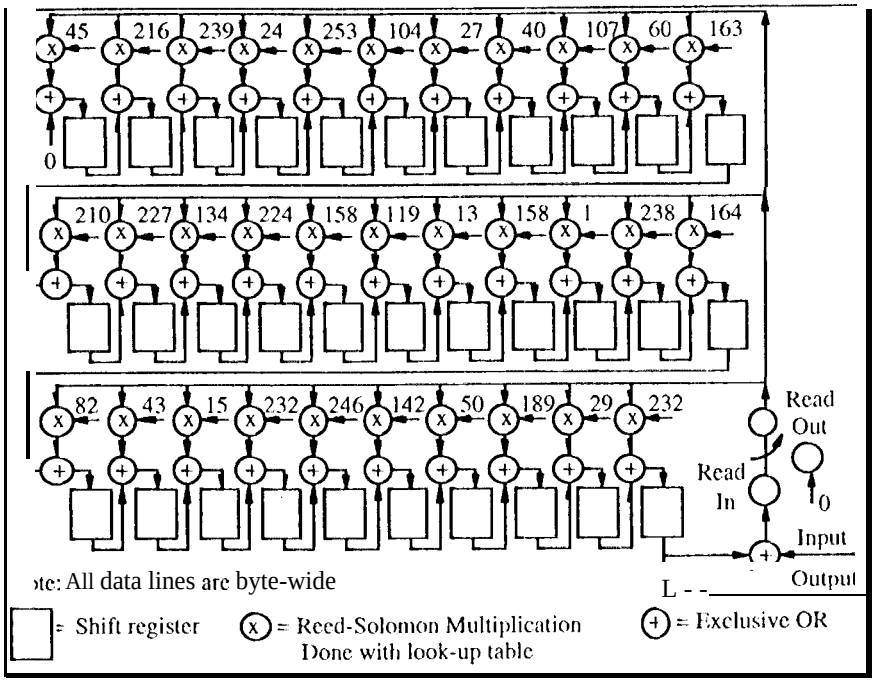

Figure 6. Reed-Solonlol\ Circuitry

After a complete telemetry frame is assembled, the bits arc passed through a convolution coder that doubles the number of hits. The 'convolutional and deconvolutional circuitry is shown as Figures 7 and 8. 'l'he diagram in Figure 9 implements the hardware circuitry shown in Figure 7. The bits arc eventually doubled again (so that each bit is present for two clock times) and used to control two voltage levels (zero anti five volts) on one channel of the double-buffered D/A converter. The other channel generates an alternating bit pattern to form the clock.

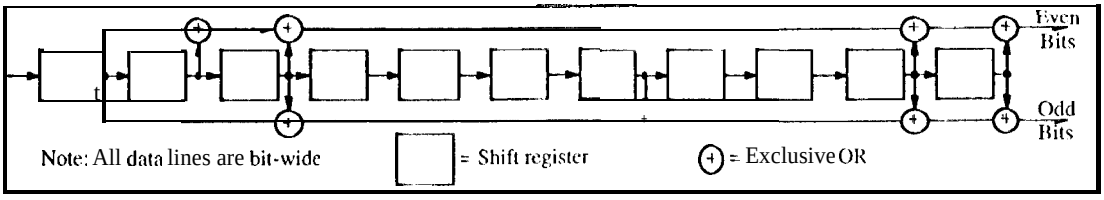

Figure 7. Convolutional Coder Circuitry

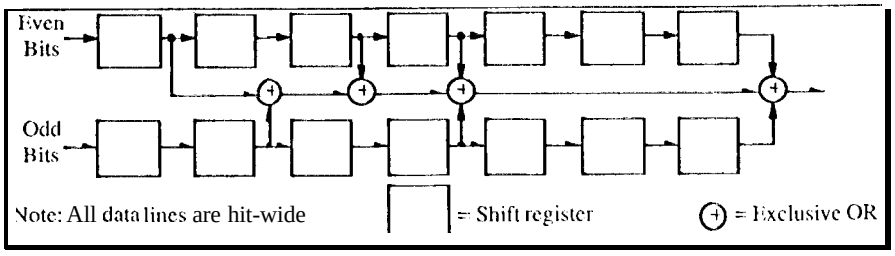

Figure 8. Convolutional Decoder Circuitry

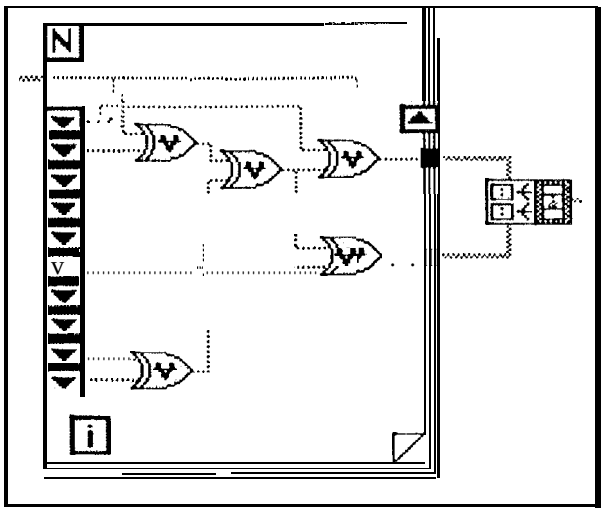

Figure 9. LabVIEW Diagram of Convolution Coder

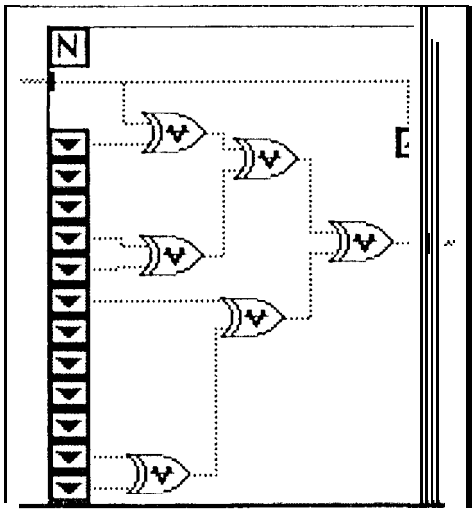

Figure 10. LabVIEW Diagram of Convolution Decoder

Telemetry Analyzer

The Telemetry Analyzer uses the double-buffered analog input with samples taken on each falling edge of the clock. Each analog sample is immediately converted to a Boolean by a simple comparison test and passed through the convolution decoder shown in Figure 10 that implements the circuitry shown in Figure 8. Higher level VI's can request any number of bits from the convolutional decoder, but normally only half the bits are used. The bits come out in two separate arrays but only one of them is significant. The correct one is the one that contains the PN Sync word so the Get Sync VI shown in Figure 11 must search for the PN Sync word in both arrays. If it finds it in the second array, it reads and discards one more bit so that the remainder of the frame is read from the first array.

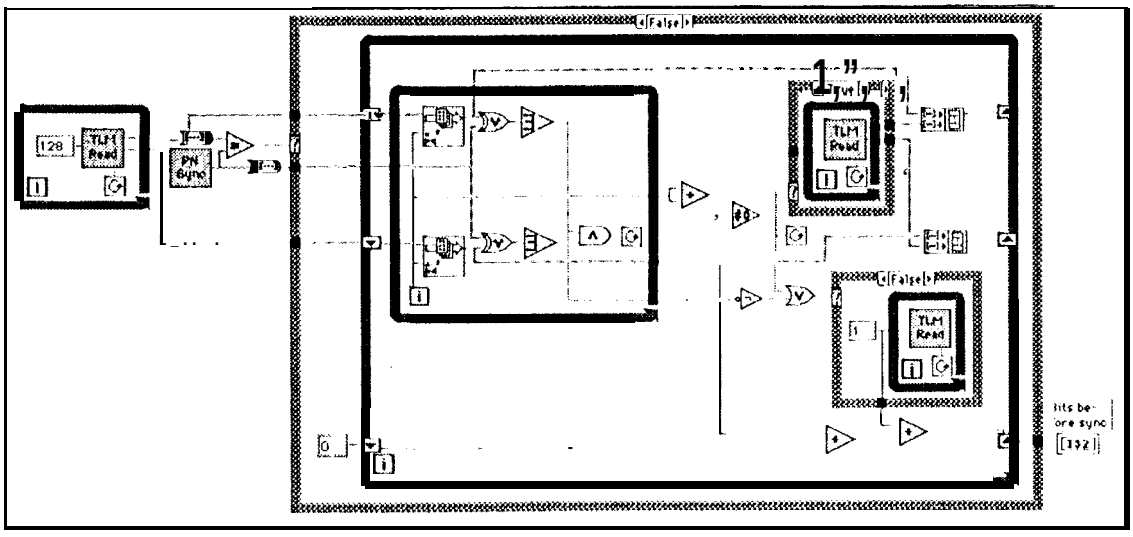

Figure 11. LabVIEW Diagram of Get Sync VI

The remainder of the Telemetry Analyzer basically performs the same functions that the Generator performs but in the opposite order. The same Reed Solomon calculations are done on each segment but instead of correcting errors they arc simply flagged on the front panel (Figure12). The header information from the packets is then stripped away and the data compared to the Predict Tables and appropriate error flags set. The VI has front panel logging turned on so that the status from each frame can be saved to disk. Each frame will have status information for eight segments including the Instrument ID, the current sequence number, the previous sequence number, error flags to indicate if the Reed-Solon]on code is incorrect, if the sequence is not properly incremented, if the packet bytes were found in the Predict '1'able but not in the proper place, and if the packet bytes were not found in the Predict l'able. The strip chart indicates which Instrument ID's each of the eight segments go with but it is not useful for data logging since only the last segment of each frame will be logged. Several other parameters relating to the frame itself arc also indicated at the top of the front panel including the frame count, the number of bits found before the PN Sync code (which should always be zero after the first frame), the current time, the actual data rate, and finally the percent margin assuming a data rate of 2000 bits pcr second. All testing was done atten times the targeted r-ate to save testing time.

Several utility VI's (Figure 13) arc available to read the previously logged data, even while logging is in progress. Among these arc a monitor to graph any of the parameters on the front panel against any of the others, e.g., to plot the bit rate against the frame count; a monitor to indicate. Instrument ID utilization in the. for-m of a bar graph; and a scrolling graph to display Instrument ID's as a function of real time.

Conclusions

With approximately eight weeks of funding over a period of three months, the visual programming effort was significantly more advanced, having gone wcii beyond the original requirements than the 'C' development effort, which did not complete the original requirements. The visual programming team worked in a very interactive mode with the customer, meeting frequently and actually 'coding' together at [hc computer. At times, coding of the task was ahead of the requirements discovery. The text-based team took the initial requirements and did not meet the customer again until their demonstration at tile end of the task.

LabVIEW was able to perform advanced data analysis tasks such as telemetry stream emulation and monitoring plus display. It proved that it is possible to usc visual programming for realistic programming applications. This task succeeded in convincing people in our organization that visual programming can significantly reduce software development time compared to text-based programming. As a result of this demonstration, additional follow-on work was awarded to the visual programming team.

Acknowledgments

The research described in this paper was carried out by the Jet Propulsion 1.aboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

1. Baroth, E. C., Hartsough, C., Johnsen, 1... McGregor, J., Powell-Meeks, M., Walsh, A., Wells, G., Chazanoff, S., and Brunzie, T. "A Survey of Data Acquisition and Analysis Software Tools, Part i," *Evaluation Engineering Magazine*, October, 1993, pp. 54-66. Part 2, November, 1993, pp. 128-140.

2. Baroth, E. C., Clark, D. J. and Losey, R. W., "Acquisition, Analysis, Control, and Visualization of Data Using Personal Computers and a Graphical-Based Programming Language," *Conference Proceedings of American Society of Engineering Educators (ASEE)*, Session 2659, Toledo, Ohio, June 2 i -2S, 1992, pp. 1447-1453.

3. Baroth, E. C., Clark, D. J. and Losey, R. W., "An Adaptive Structure Data Acquisition System using a Graphical-Based Programming Language," *Conference Proceedings of Fourth A IA A/A ir Force/NASA/OA I Symposium on Multidisciplinary Analysis and Optimization*, AIAA-92-4833-CP, Cleveland, Ohio, September 2 I -23, i 992, pp. 1104 1110.

4. Wells, G. and Baroth, E. C., "Telemetry Monitoring and Display using LabVIEW," *Proceedings of National Instruments User Symposium*, Austin, Texas, March 28-30, 1993.

5. Bulkeley, D., "Today's Equipment Tests Tomorrow's Designs," *Design News Magazine*, May 17, 1993, pp. 82-86.

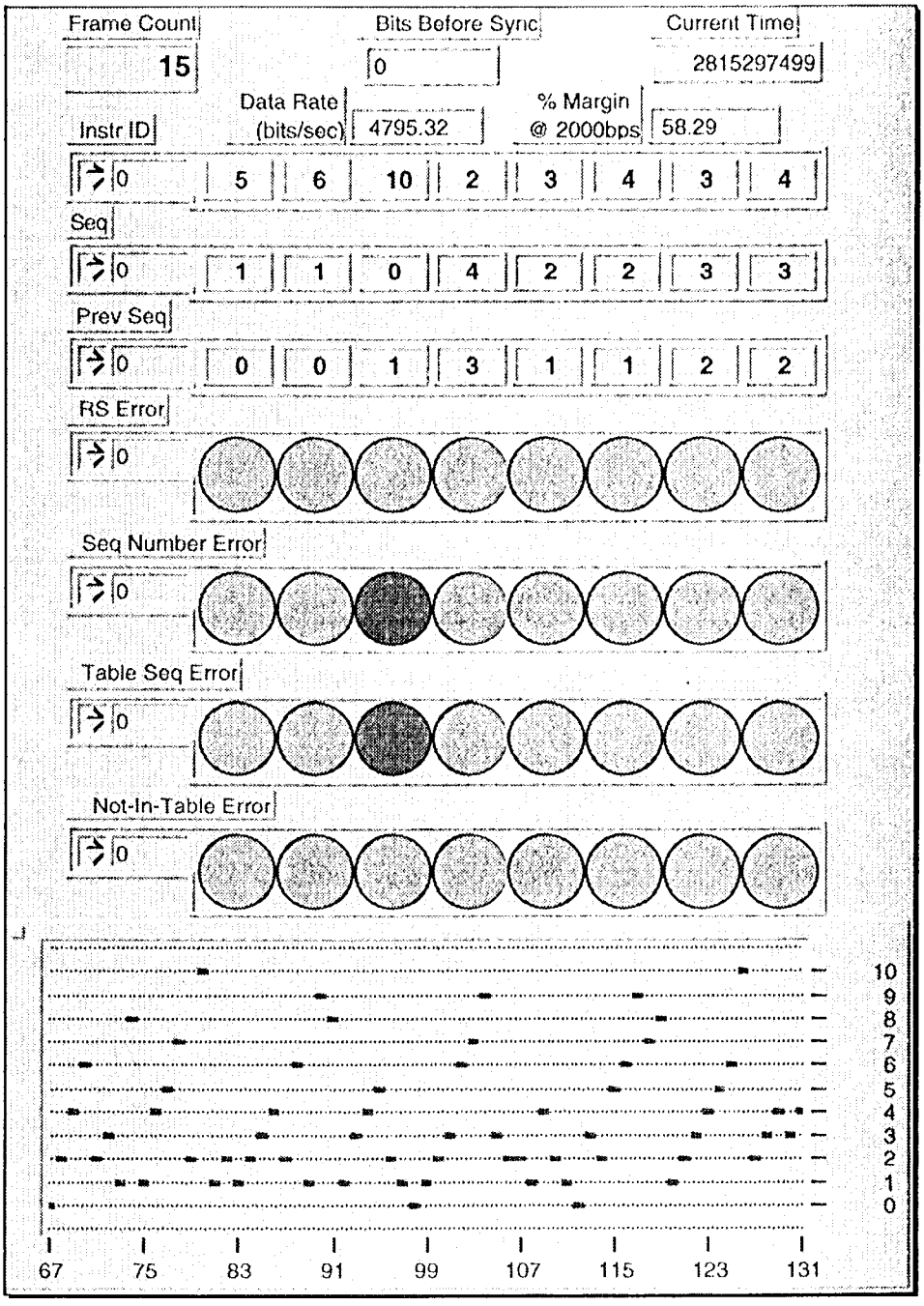

Figure 12. Telemetry Analyzer User Interface (LabVIEW Front Panel)

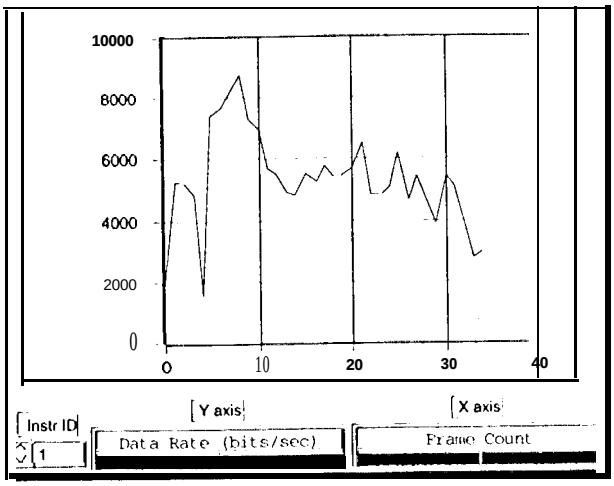

Figure 13a. Bit Rate vs. Frame Count

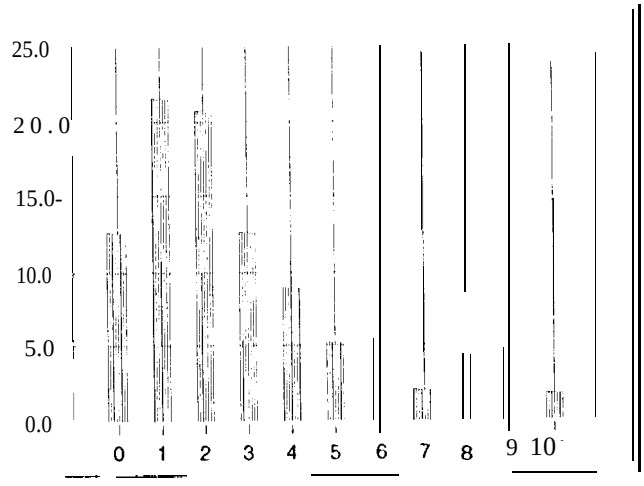

Figure 13h. InstrumentID Utilization

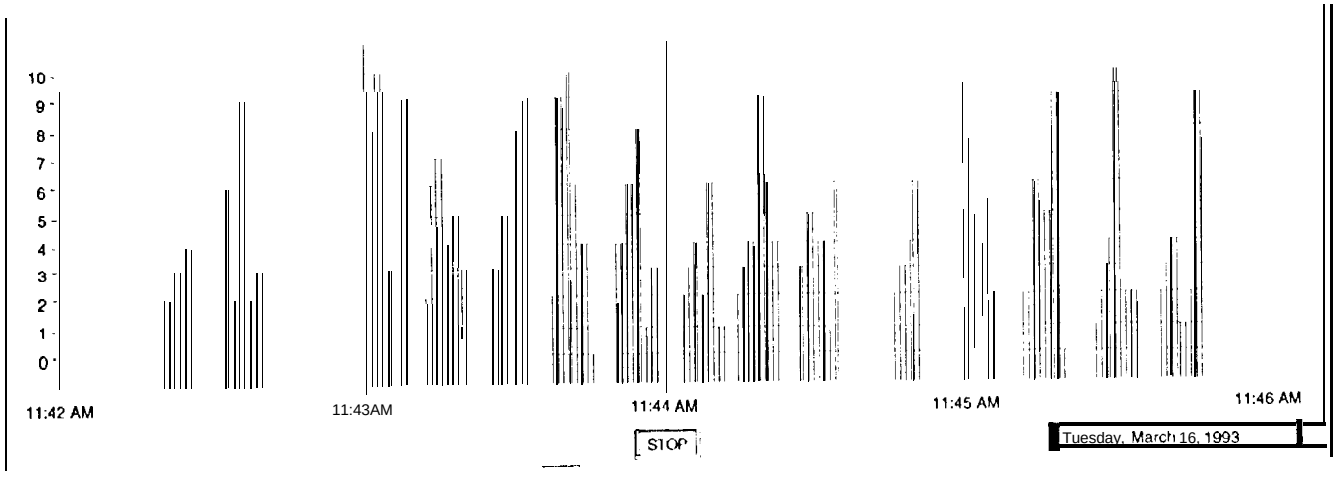

Figure 13c. InstrumentID vs. Real-Tiau2

6. Puttre´, M., "Software Makes Its Home in the Lab," *Mechanical Engineering Magazine*, October, 1992, pp. 75-78.

7. Kent, G., "Automated RF Test System for Digital Cellular Telephones," *Proceedings from NEPCON West '93*, Anaheim, California, February 7-11, 1993, pp. 1055-1064.

8. Henderson, J. R., "Sequential File Creation for Automated Test Procedures," *Proceedings from NEPCON West '93*, Anaheim, California, February 7-11, 1993, pp. 1065-1077.

9. Jordan, S. C., "Cutting Costs the Old Fashioned Way," *Proceedings from NEPCON West '93*, Anaheim, California. February 7-11, 1993, pp. 1921-1931.

10. Breeman, D., "Jet Repulsion Lab Aids in Space Craft Project," *Scientific Computing and Automation*, November, 1993, pp. 26-28.